Schedulability Analysis of periodic task of uniprocessor system on Real Time System

ANJANA TUDU, 109CS0196
Department of Computer Science & Engineering
National Institute of Technology Rourkela
Rourkela-769 008, Orissa, INDIA
2013

Schedulability Analysis of periodic task of uniprocessor system on Real Time System

Thesis submitted in partial fulfillment
of the requirements for the degree of

BACHELOR OF TECHNOLOGY

by

Anjana Tudu,109cs0196

Thesis Advisor: Bibhudatta Sahoo

National Institute of Technology Rourkela
Department of Computer Science & Engineering
Rourkela-769 008, Orissa, INDIA

To my Parents

Department of Computer Science and Engineering National Institute of Technology Rourkela

Rourkela-769 008, Orissa, India.

Certificate

This is to certify that the work in the thesis entitled Schedulability Analysis of periodic task of uniprocessor system on Real time System submitted by Miss. **Anjana Tudu** in partial fulfilment of the requirements for th**e Award of the degree of Bachelor** of Technology in Computer Science and Engineering, **2009– 2013** in the department of Computer Science and Engineering, National Institute of Technology, Rourkela (Deemed University) is an authentic work carried out by her under my supervision and guidance.

Date:
Place:

Prof. **Bibhudatta Sahoo**
NIT Rourkela
Department of Computer Science & Engg.
National Institute of Technology
Rourkela-769008

Acknowledgment

My first thanks are to the Almighty, without whose blessings I wouldn't have been writing this "acknowledgments". I then would like to express my heartfelt thanks to my supervisor, Prof Bibhudatta Sahoo,CSE department for his guidance, support, and encouragement during the course of my bachelor study at the National Institute of Technology, Rourkela. I am especially indebted to him for teaching me both research and writing skills, which have been proven beneficial for my current research and future career. Without his endless efforts, knowledge, patience, and answers to my numerous questions, this research would have never been possible. The experimental methods and results presented in this thesis have been influenced by him in one way or the other. It has been a great honor and pleasure for me to do research under.

I am also thankful to Prof. A.K. Turuk, Prof. B.Majhi, Prof. P. M. Khillar , Prof. D P Mahapatra,, Prof S.K.Rath,Prof. Suchismita Chinara mam,Prof K.Sathya Babu, Prof. Manmath supervision for giving encouragement during my thesis work.

I thank my friends & all the members of the Department of Computer Science and Engineering, and the Institute, who helped me by providing the necessary resources, and in various other ways, in the completion of my work.

Finally, I want to dedicate this thesis to my parents, my family members and my beloved one for their unlimited support and strength. Without their dedication and dependability, I could not have pursued my BTech. degree at the National Institute of Technology Rourkela.

Anjana Tudu

Roll No: 109cs0196

Schedulability Analysis of periodic task of uniprocessor system on Real Time System

Anjana Tudu
NIT Rourkela

Abstract

A real time system is a system that must satisfy explicit bounded response-time constraints, otherwise risk severe consequences including failure. Failure happens when a system cannot satisfy one or more of the requirements laid out in the formal system specification. The problem of real-time scheduling spans a broad spectrum of algorithms from simple uniprocessor to highly sophisticated multiprocessor scheduling algorithms. In this project, we will study the characteristics and constraints of real-time tasks which should be scheduled to be executed. Analysis methods and the concept of optimality criteria, which leads to design appropriate scheduling algorithms, will also be addressed. Then, we study real-time scheduling algorithms for uniprocessor systems, which can be divided into two major classes: off-line and on-line. On-line algorithms are partitioned into either static or dynamic-priority based algorithms. We will observe both preemptive and non-preemptive static-priority based algorithms. For dynamic-priority based algorithms, we study the two subsets; namely, planning based and best effort scheduling algorithms. This project compares RM against EDF under several aspects, using existing theoretical results, specific simulation experiments, or simple counter examples to show that many common beliefs are either false or only restricted to specific situations.

Contents

LIST OF FIGURES PAGE NO

CHAPTER 1: INTRODUCTION

1.1 INTRODUCTION

In this materialistic world, the purpose of a real-time system is to have a physical effect within a chosen time-limit. A real-time system composed of a controlling system (computer) and a controlled system (environment). The computer interacts with its environment based on information available about the surroundings. A real-time computer, controls a device or process, sensors provide readings at periodic intervals and the computer responds by sending signals to actuators. There may be unknowing events and they must also receive a response. In all cases, there is a time bound within which the response should be delivered. The potential of the computer to meet these bound demands depends on its competence to perform the necessary computations within the given time. If a number of events simultaneously occur eventually, the computer will need to schedule the computations so that each response is recorded within the required time bounds. It may happen that, the system is unable to meet all the possible abrupt demands. In this situation we say that the system lacks sufficient resources; a system with unlimited resources and capable of processing at infinite speed can satisfy any such timing constraint. Failure to meet these timing constraint for a response can result into different consequences; there may be no effect at all, or the effects may be small or correctable, or the results may be catastrophic ruin. Each task occurring in a real-time system has some timing properties. These timing properties should be considered when scheduling tasks on a real-time system. **[1,10,11]**

- *Release time* (or *ready time*): Time at which the task is ready for processing.
- *Deadline*: Time by which implementation of the task should be completed, after the task is released.
- *Minimum delay*: Minimum required time that must pass before the execution of the task is started, after the task is released.

- *Maximum delay*: Maximum amount of time allowed to pass before the execution of the task is started, after the task is released.
- *Worst case execution time*: Maximum time taken to complete the task, after the task is released. The worst case execution time is also referred to as the *worst case response time*.
- *Run time*: Time taken to complete the task without break, after the task is released.
- *Weight* (or *priority*): Relative urgency of the task**.[1]**

The objective of a computer controller might be to command the robots to move parts from machines to conveyors in some required fashion without colliding with other objects. If the computer controlling a robot does not command it to stop or turn in time, the robot might collide with another object on the factory floor .A real-time system will usually have to meet many demands within a bound time. The significance of the demands may vary with their nature (e.g. a safety-related demand may be more important than a simple data-logging demand) or with the time available for a response. So the allocation of the system resources needs to be planned so that all demands are met by the time of their respective deadlines.

The scheduling is done using a scheduler which implements a scheduling policy that defines how the resources of the system are allocated to the program. Scheduling policies are revealed mathematically so the accuracy of the formal specification and program development stages can be complemented by a mathematical timing analysis of the program properties**.[10,16,2]**

1.2 LITERATURE REVIEW

In many application domains there is use of Real time uniprocessor computing systems and the availability of a continuous service is very important.Arezou Mohammadi and Selim G.Akl, [1], 2005, outlines the study of real time scheduling algorithms for uniprocessor system which have been divided into two algorithm: static and dynamic. The importance of predictable scheduling in hard-real time computing systems has been shown by G.C Buttazzo[2],2005.

Nasro Min-Allah,Samee Ullah Khan,[3],2004,comparitively studied about rate monotonic schedulability.J.Goossens and P.Richard,[4],2004,had addressed the problem of runtime monitering the real hard time programs in the runtime monitering community.From the results shown by Liu and Layland on RMS and EDF algorithms, comparison of the two algorithm is done by Giorgio C.Buttazzo[5],2003 .

James H. Anderson [6], 2003 contributed a new EDF-based scheme that ensures bounded deadline tardiness. In this scheme, per-task utilizations must be capped, but overall utilization need not be restricted. Nasro Min-Allah · Samee Ullah Khan, Nasir Ghani. Juan Li. LizheWang ·Pascal Bouvry [8],2001 assist the system designers in the process of selecting a suitable technique from the existing schedulability test.Steve Schneider[9],2000 approach has been widely used in the specification, analysis and verification of concurrent and real time systems, and for understanding the particular issues that can arise when concurrency is present. C. M. Krishna and K. G. Shin [10], 1997 analyze some of the popular real-time operating systems and investigate why these popular systems cannot be used across all applications. We also examine the POSIX standards for RTOS and their implications.

1.3 MOTIVATION

A "Schedulability Analysis" is one that is always performed at desired level of service periodic EDF and RMS components that constitute the system. This analysis is required in systems such as telephone system, banking systems, railway system, airport systems, stock market,atm,hospital(pacemaker),nuclear reactor control system where reliability is ensured.So,the problem in this schedulability analysis has been put forwarded, which are shown here for system for periodic tasks in real time system uniprocessor platform**.[10]**

1.4 PROBLEM STATEMENT

Schedulability analysis of EDF and RMS is a fundamental aspect of building uniprocessor system, which constitutes a major part of building a system. To ensure the correctness of a system, this analysis are not only tested for functional correctness but also for timeliness. A set of n periodic tasks are to schedule on single processor where CPU utilization is calculated. The following parameters are calculated and discussed showing that they are either false or only restricted to specific situation.

They are Implementation Complexity, Runtime Overload, Scheduling Analysis, and Robustness during Overload, Jitter and Latency.

CHAPTERT 2: REAL-TIME-SYSTEM: EDF and RMS

Real-time systems are defined as those systems in which the correctness of the system depends not only on the logical solution of result, but also on the time at which the results are shown. If the timing limitations of the system are not been handled, system failure is surely going to happen. Hence, it is necessary that the timing constraints of the system are assured to be met. Assuring timing behaviour need that the system to be *predictable*. Predictability means that when a job is activated it should be possible to determine its completion time with surety. It is also required that the system attain a high degree of utilization while satisfying the timing constraints of the system. [**16, 11, 10, 2, 3]**It is imperative that the state of the environment, as got by the controlling system, be reliable with the real state of the environment. Otherwise, the results of the controlling three systems' activities may be terrible. Therefore, regular periodic monitoring of the environment as well as timely processing of the sensed information is necessary**.[16,10]**

A real-time application is basically composed of multiple jobs or tasks with different levels of cruciality. Although missing deadlines is not happening in a real-time system, *soft real-time tasks* are those which miss some deadlines and the system could still work error free. But, missing some deadlines for soft real-time tasks will lead to pay consequences. *Hard real time tasks* cannot miss any such deadline; if does, catastrophic or fatal results will be produced in the system. There exists also one more group of real-time tasks, called *firm real-time tasks*, which are such that the faster they finish their computations before their deadlines, the more rewards they gain **.[10,16,11]**

2.1 UNIPROCESSOR REAL TIME SCHEDULING

It decide when and where to execute tasks such that-Time requirements are meet, Performance /resource usage is optimized. Scheduling Analysis is the study the properties of scheduling policies. Can a task set meet the timing requirement with certain given scheduling policies? Scheduling Synthesis for a given task set, is what scheduling algorithm produces a feasible schedule? Feasible Schedule: Each task must reach its deadline without violating any constraints. Optimal Schedule: Optimality criterion assesses the relative merit of competing feasible schedules.

Two most important priorities driven preemption scheduling schemes are:

(i) Rate Monotonic Scheduling (RMS)

(ii) Earliest Deadline First (EDF)

Scheduling Test: - A schedulability test is used to validate that a given application can satisfy its specified deadlines when scheduled according to a specific scheduling algorithm.

Schedulability Utilization: - A schedulable utilization is the maximum utilization allowed for a set of tasks that will guarantee a feasible scheduling of this task set.

There are mainly two types of important schedulers.

- Compile-time (static)
- Run-time (on-line or dynamic)

Optimal Scheduler: - An optimal scheduler is one which may fail to meet a deadline of a task only if no other scheduler can.

Optimal means fastest average response time / shortest average waiting time.

Parameters of the task Ti are:

s : start,release,ready or arrival time

c: (maximum) computation time

d: relative deadline (deadline relative to the task's start time)

D: absolute deadline (wall clock time deadline)

There are three main types of tasks.

1. A single-instance task execute only once.
2. A sporadic task has zero or more instance.
3. An aperiodic task is a sporadic task with either a soft deadline or no deadline.

If the task has more than one instance, we have p: period (for periodic tasks): minimum separation.

Additional constraints are -frequency of tasks requesting service periodically, precedence relations among tasks and subtasks, resources shared by tasks, whether task preemption is allowed or not.

A task = (C, T). C: worst case execution time/compiling time(C<=T!),T: period (D=T)

A task set: (Ci, Ti)

All tasks are independent. The periods of the task start at 0 simultaneously.

CPU Utilization

C/T is the CPU utilization of a task. $\cup = \frac{C_i}{T_i}$ is the CPU utilization of a task set.CPU utilization is a measure on how busy the processor could be during the shortest repeating cycle $T_1 * T_2 * T_3 \ldots . * T_n$.[1]

- $\cup > 1$ (overload) : Some job or task will failed to meet its deadline no matter what algorithm you use.
- $\cup \leq 1$ It will depend on the scheduling algorithms.
- If $\cup = 1$ and the CPU is kept busy (non idle algorithm eg: EDF) all deadline will be meet.

2.2 A Simple Model

Let us consider a simple real-time system containing a periodic hard real-time task which should be processed on one processor **[10]**. The task does not require any extra resource. The priority of the task is fixed.

We define a simple real-time program as follows: Program H receives an event from a sensor every P units of time (i.e. the *inter-arrival time* is P). A task is defined as the processing of an event. In the worst case the task requires C units of computation time. The execution of the task should be completed by D time units after the task starts. If D<C, the deadline cannot be accomplished. If P<D, the program must still process each event in a time >P if no events are found to be lost Therefore the deadline is effectively bounded by P and we need to handle only those cases where C≤D≤P. [**16, 10, 11]**

Now consider a program which receives events from *two* sensors. Inputs from Sensor 1 come every P1 time units and each needs C1 time units for computation; events from Sensor 2 come every P2 time units and each needs C2 time span units. Let the deadlines are the equal as the periods, that is P1 time units for Sensor 1 and P2 time units for Sensor 2. Under what situations or condition will these deadlines be accomplished? More often, if a program receives actions from n such devices, how can it can satisfactorily determine the deadline for each device?

Before we begin to analyse this problem statement, we first know our assumptions as follows. We assume that the real-time program consists of a number of *independent tasks* that do not share data with each other. Also, we take as that each task is periodically invoked by the occurrence of a particular event **[10, 11]**. The system has one processor; the system

periodically receives events from the external environment and these are unbuffered. Each action is an invocation for only a particular task. The events may be periodically produced by the environment or the system may have a timer that periodically creates the events. The processor is idle when it is not executing a task.

Let the tasks of program H beT_1,T_2,T_3....T_n.Let the inter-arrival timer, or *period*, for T_i invocation to task P_i be and the computation time for such invocation be C_i

2.2.1 Scheduling for the Simple Model

One way to schedule the program is to analyse its tasks *statically* and determine their timing constraints properties. These times can be used to create a *fixed scheduling* table according to which tasks will be dispatched for execution at run-time **[2,9, 12]**. Thus, the order of execution of tasks is fixed and it is assumed that their execution times are also fixed.
Typically, if tasksT_1,T_2,T_3....T_n have periodsP_1,P_2,P_3.....P_n.The table must cover scheduling for length of time equal to the *least common multiple* of the periods, i.e. lcm {P_1,P_2,P_3.....P_n},as that is the time in which each task will have an integral number of invocations.

If any of the P_i are co-primes, this length of time can be enormously large so where possible it is advisable to choose values of that are small multiples of common value..Lets define a *hyper-period* as the period same to the least common multiple of the periodsP_1,P_2,P_3.....$P_n$of the n periodic tasks.

Static scheduling has the significant advantage that the order of execution of tasks is determined *off-line* (before the run of the program), so the run-time scheduling expenses can be very little. In the meanwhile it has some major disadvantages**[4].**

In scheduling property, a *priority* is basically a positive integer representing the importance assigned to an task. By convention, the importance is in opposite order to the numeric value of the priority, and we know that priority 1 is the highest level of priority. We have to assume here that a task is having a single, fixed priority. Lets consider the following two simple priority scheduling disciplines:

Non-preemptive based execution: When the processor is found to be idle, the ready task with the highest priority is chosen for execution; and once chosen, a task has to run to completion.

Preemptive based execution: When the processor is found idle, the ready task with the highest priority is chosen for implementation at any time, execution of that task can be pre-empted if a task of higher priority becomes ready. Then, at all times the processor is found idle or executing the ready task with the highest priority**.[5]**

2.3 RATE MONOTONIC SCHEDULING

Method of assigning priorities to a set of processes or assigning priorities as a monotonic function of the rate of a (periodic) process. Rate monotonic scheduling provides simple inequality comparing total processor utilization to a theoretically determined bound-that serves as a sufficient condition to ensure that all processes will complete their work by the end of their periods.

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{1/n} - 1\right)$$

Where C_i = the execution time, T_i = Period associated with periodic task.

For this we have Utilization Bound (UB) Test.

1. It has three possible outcomes:
   - 0 ≤U≤ U(n) -Success
   - U(n) < U ≤1.00 -Inclusive
   - 1.000 < U - Overload
2. Draw Time line
3. Schedulability: CT Test

Theorem: - For a given set of independent, periodic tasks if each task has to meet its first deadline, with worst-case task phasing, the deadline will always be meet.

Completion Time Test:

Tasks suffer interference from higher priority tasks. Response time is the time that passes since the task is released and until it finishes.

$$R_i = \quad C_i + I_i \ , R_i = C_i + \sum_{j \in hp(i)} \left(\frac{R_i}{T_j}\right) C_j$$

Let W_i = completion time of task W_i may be completed by the following iterative formula:

$$W_i^{n+1} = C_i + \sum_{j \in hp(i)} \left(\frac{W_i^n}{T_j}\right) C_j \qquad \text{Where } W_i(0) = 0$$

Task is schedule if its completion time is before its deadline. That is $W_i \leq T_i$.

Rate Monotonic Scheduling: Task Model

Assume a set of periodic tasks: (C_i, T_i)

- D_i=T_i
- Task is always released at the start of their periods.
- Task is independent.

Rate fixed/Static-priority scheduling

- Rate Monotonic fixed-priority assignment are those:[1,2]

  Higher priorities are given to tasks with small periods.
- Run-time Scheduling are those:

  Preemptive with highest priority first
- RMS is optimal in the same ;

  If a task is schedulable with any fixed-priority scheduling algorithm, it is schedulable with RMS.

RMS: Schedulability Test

- U < 1 doesn't imply "schedulable" with RMS.
- Utilization bound: given a task set S, find X(S) such that U ≤ X(S) if and only if S is schedulable by RMS (necessary and sufficient test).The bound X(S) for EDF is 1.
- The famous Utilization Bound Test (UB test)[by Liu and Layland , 1973 : a classic result]

  *Assume a set of n independent tasks: S = $\{(C_1, T_1),(C_2, T_2)\ldots.(C_n, T_n)\}$ and

  U= 1

  ***** If U ≤n*($2^{\frac{1}{n}} - 1$), then S is schedulable by RMS.

  ***** Here the bound depends completely on the size of the task set.

Schedulability Test 1: Given a set of n independent, preempt able and periodic tasks on a uniprocessor such that their relative deadline are equal to or larger than their respective periods and that their periods are exact multiples of each other. If u is the total utilization of this task set.For feasible scheduling of this task set has a necessary condition to be followed:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1$$ **[1]**

Schedulability Test 2: Given a set of n independent, preamble and periodic tasks on a uniprocessor, let U be the total utilization of this task set. A sufficient condition for feasible schedulability of this task set is

$$U \leq n.\left(2^{1/n} - 1\right)$$: Exception cases are also there

Schedulability Test 3:- Let

$$w_i(t) = \sum_{k=1}^{i} c_k \left\lceil \frac{t}{p_k} \right\rceil, 0< t \leq p_i$$

The following inequality: $w_i(t) \leq t$ holds for any time instant t chosen as follows t= kp_j , j=1,….,i,k= 1, ...,$\left\lfloor \frac{p_i}{p_j} \right\rfloor$ .If task j_i is RM-schedulable. If di not equal to pi by min (d_i,p_i) in the expression.

2.3.1 RMS ALGORITHM [3]

Algorithm Response time analysis

Procedure b Feasible RTI (tasks-vector)

1. **int n ;**
2. **n= size(tasks-vector,1);**
3. **bFeasible = 1;**
4. **int R = 0; Rold = 0;**
5. **for all i=1:n do**
6. **R = R+ tasks-vector(i,1);**
7. **while (R > Rold)**
8. **Rold = R;**
9. **R = tasks-vector(i,1);**
10. **for all k=1:i-1 do**
11. **R = R + ceil(Rold/tasks-vector(k,2)) × tasks-vector(k,1);**
12. **end for**
13. **if (R > tasks-vector(i,2)) then**
14. **bFeasible = 0;**
15. **break;**
16. **end if**
17. **end for**
18. **if (bFeasible == 0) then**
19. **break;**
20. **end if**
21. **end-while**
22. **end function**

Summary: Let's note three ways to check Schedulability

1. UB test (simple but conservative)

2. Response time calculation (precise test)
3. For the first periods construct a schedule.
    - Let's assume that at time 0 the first instances arrive.(critical instant)
    - Then draw the schedule for the first periods.
    - Check if all tasks are finished before the end of the first periods, schedulable, otherwise NO.

4. RMS for task with $D \leq$T
    - RMS is no longer optimal.
    - Utilization bound test has to be modified.
    - Response time test is still applicable. Assuming that fixed-priority assignment is adopted. But considering the critical instant and checking the first deadline principle are still applicable.

2.4 EARLIER DEADLINE FIRST (EDF)

- Task model: a set of independent periodic tasks.
- EDF: Whenever a new task arrives, sort the ready queue so that the task closest to the end of its period assigned the highest priority. Preempt the running task if it is not placed in the first of the queue in the last sorting.
- EDF is optimal – EDF can schedule the task set if anyone else can
- Example: Task set: {(2, 5), (4, 7)}, U=2/5+4/7=34/35 = 0.47(approx.) is schedulable.
- EDF is a deadline monotonic (DM) scheduling algorithm.

Schedulability Test 4:- Let ci denote the computation time a task Ji. For a set of n periodic tasks such that the relative deadline di of each task is equal to or greater than its respective period pi ($d_i \leq p_i$) , a necessary and sufficient condition for feasible schedulability of this task set on a uniprocessor is that the utilization of the tasks is than or equal to 1.

$$\mathrm{U} = \sum_{i=1}^{n} \frac{c_i}{p_i} \leq 1$$

Schedulability Test 5: A sufficient condition for feasible schedulability of a set of independent, preemptable and periodic task on a processor is

$$\sum_{i=1}^{n} \frac{c_i}{\min(d_i, p_i)} \leq 1$$

If di=pi then Schedulability Test 5 is same as 4.

Schedulability Test 6: Given a set of n periodic independent, preemptable tasks on a uniprocessor. Let U be the utilization as defined in Schedulability test 4.dmax be the maximum relative deadline among multiple LCM of the tasks periods and s(t) be the sum of the

computation times of the task with absolute deadlines less than t. This task set is not EDF-schedulable if either of the following conditions is true:

$$U > 1$$

$$\exists t < min\left(P + d_{max}\left(\frac{U}{1-U}\right)\max_{1\le i\le n}(p_i - d_i)\right)$$

Such that s (t)>t **[1]**

Sporadic Tasks: - It may be released at any time instant but a minimum separation exists between releases of consecutive instances of the same sporadic task.

A simple approach to schedule to schedule tasks is to treat them as periodic tasks with the minimum separation times as their periods. Then we schedule the periodic equivalents of these sporadic tasks using the scheduling algorithm. The second approach to schedule sporadic tasks is to treat them as one periodic task is with the highest priority and a period chosen to accommodate the minimum separates and computation requirements of this collection of sporadic task.

Schedulability Test 7:- LetP_s and C_s be the period and allocated time for the deferred server. Let U_s=c_i/p_i be the utilization of the server. A set of n independent preemptable and periodic task with relative deadline the same as the corresponding periods on a uniprocessor such that the period satisfy p_s<p_1<p_2<……<p_n<2p_s and p_n> p_s+c_s is RM schedulable if the total utilization this task set (including the DS) is at most.

$$\cup(n)=(n-1)\left[\left(\frac{U_s+2}{U_s+1}\right)^{\frac{1}{n-1}}-1\right] \quad \textbf{[2]}$$

Scheduling NonPreemptive Tasks – Sporadic tasks

We apply the schedulability strategies for sporadic tasks introduced earlier by first transferring the sporadic tasks into equivalent periodic tasks yielding to schedulability test.

Schedulability Test 8: Suppose we have a set M of tasks that is the union of a set Mp of periodic tasks. Let the nominal laxity (or initial) L_i of task $T_i$be di-ci .Each sporadic task T_i= (c_i, d_i, p_i) is replaced by an equivalent periodic task T_i'= (c_i',d_i',p_i') as follows:

$$c_i' = c_i$$
$$p_i' = min(p_i, l_i + 1)$$
$$d_i' = c_i$$

A sporadic task (c, d, p) can be transferred into and scheduled as a periodic task (c', d', p') if the condition

(i) d'≥d≥c (ii) c'=c (iii) p' ≥ d-d'+1 **[10]**

2.4.1 ALGORTIHM EDF [6]

Global variable:

u : array [1…N] of double initially 0.0;

s : array [1…N][1…M] of double initially 0.0;

p : array [1…N][1…2] of 0…M initially 0;

m: array [1…M][1…N] of 0…N initially 0;

f: array [1…M][1…N] of 0…N initially 0;

Local variable:

Proc: 1…M initially 1; // Tasks and processors are both considered sequentially.//

Task: 1…N;

AvailUtil: double;

mt, ft; integer initially 0

```
1.  AvailUtil := 1.0;
2.  For task := 1 to n do
3.      If AvailUtil >= u[task] then
4.          s[task][proc] := Availutil – u[task];  // tasks are assigned to proc as long as the
            processing capacity of proc is not exhausted.//
5.          AvailUtil := AvailUtil – u[task];
6.          ft := ft + 1;
7.          P[task][1] := proc;
8.          f[proc][ft] := task;
9.      else
10.         If AvailUtil > 0 then
11.         s[task][proc] := AvailUtil;
12.         mt = mt + 1;
13.         m[proc][mt] := task;
14.         p[task][1] > p[task][2] := proc,proc + 1;
15.         mt,ft := 0,1;
16.         m[proc + 1][mt] := task
17.     else
18.         mt,ft := 0,1;
19.         p[task][1] := proc +1;
20.         f[proc + 1][ft] := task;
21.     fi
22.         proc := proc +1;
23.         s[task][proc] := u[task] – s[task][proc -1];
24.         AvailUtil := 1 – s[task][proc];
25.     fi
```

2.5 Quality of Service of RMS Scheduling

The RM scheduling algorithm is one of the most widely studied and used in practice. It is a uniprocessor static-priority preemptive scheme. For the RM scheduling algorithm, in addition to assumptions (**a**) to (**c**), we assume that all tasks are periodic and the priority of task τ_i is higher than the priority of task τ_j, where i<j. The RM scheduling algorithm

is an example of priority driven algorithms with static priority assignment in the sense that the priorities of all instances are known even before their arrival. The priorities of all instances of each task are the equal. They are found only by the period of the task. A periodic task consists of an infinite sequence of instances with periodic ready times, where the deadline of a request could be greater than, less than, or equal to the ready time of the prospering instance. Further, the execution times for all the instances of a task remains equal. A periodic task $\tau_i$is characterized by three parametersP_i, the period of the instancec_i, the execution time, and D_i, the deadline of the tasks. The utilization factor of a set of periodic tasks is defined by $\sum_{i=1}^{n} \frac{C_i}{P_i}$, where $P_1, P_2, P_3 \dots \dots P_n$ are the periods and are the execution times of the n tasks. If $\sum_{i=1}^{n} {C_i}/{P_i} \leq n\left(2^{1/n} - 1\right)$ where n is the number of tasks to be scheduled, then the Rate Monotonic algorithm will schedule all the given tasks before they meet their respective deadlines. Here this is a sufficient, but not a compulsory, condition. That is, there may be task sets with utilization greater than $n\left(2^{1/n} - 1\right)$that are schedulable by the RM algorithm.

A given set of tasks is said to be RM-schedulable if the RM algorithm produces a schedule that meets all the commitments or deadlines. The sufficient and necessary conditions for feasibility of RM scheduling are studied as follows**.[12]**

Given a set of n periodic tasks $\tau_1, \tau_2, \tau_3, \dots . \tau_n$ whose periods and execution times are $P_1, P_2, P_3, \dots P_n$and $C_1, C_2, C_3, \dots . C_n$ respectively, we suppose task $\tau_i$completes executing at t. We consider the following notation:

$$W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{P_j} \right\rceil = t - idle\ time$$

$$L_i(t) = \frac{W_i(t)}{t}$$

$$\mathrm{L} = \min_{0 \leq t \leq P_1} L_i(t)$$

Task τ_i can be feasibly scheduled using RM *if and only if* $L_i(t) \leq 1$. In this case $\tau_1, \tau_2, \dots . \tau_{i-1}$are also feasibly scheduled. Thus far, we have only considered periodic tasks. As the sporadic tasks are irregularly released, that is often in response to some task in the operating environment. While sporadic tasks do not have periods associated with them, there must be some maximum rate at which they can be released. That is, we must have some minimum inter arrival time between the release time of successive iterations of sporadic tasks. Otherwise, there is no limit to the amount of workload that sporadic tasks can add to the system and it will be impossible to guarantee that deadlines are met**.[13]**

One drawback of the RM algorithm is that task priorities are defined by their periods. Sometimes, we must change the task priorities to ensure that all critical tasks get completed. Suppose that we are given a set of tasks containing two tasks $\tau_i$andτ_j, where$P_i < P_j$, but τ_j is a critical task and τ_i is a noncritical task. We will check the feasibility of the Rate Monotonic scheduling algorithm for the tasks$\tau_1, \tau_2, \tau_3, \ldots . \tau_n$. Suppose that if we take the worst-case execution times of the tasks, we cannot guarantee the schedulability of the tasks. However, in the average case, they are all Rate Monotonic schedulable. The problem is to arrange matters so that all the critical tasks can meet their deadlines under the RM algorithm even in the worst case, while the noncritical tasks, such as τ_i, meet their deadlines in all other cases. The solution follows any two methods under given..

We lengthen the period of the noncritical task, i.e.τ_i, by a factor of k. The original task should also be replaced by tasks, where each is phased by the correct amount. The parameter k should be chosen such that we obtain $P_i' > P_j$

We reduce the period of the critical task, i.e.τ_j, by a factor of k . Then we should replace the original task by one whose (both worst case and average case) execution time is also reduced by a factor k. The parameter k has to be chosen such that we obtain $P_i' > P_j$.

So far, we have assumed that the relative deadline of a task is equal to its period. If we relax this assumption, the RM algorithm is no longer an optimum static-priority scheduling algorithm. When $D_i \leq P_i$, at most one initiation of the same task can be alive at any one time. However, when $D_i > P_i$, it is possible for multiple initiations of the same task to be alive instantly. For the later case, we will check a number of initiations to get the worst-case response time. Thus, checking for Rate Monotonic-schedulability for the case $D_i > P_i$ is much harder than for the case $D_i \leq P_i$,.Suppose we have a task set for which there exists a γ such that $D_i = {}^{\gamma}P_i$, for each task τ_i. In, the necessary and sufficient condition for the tasks of the set to be RM-schedulable is given. The RM algorithm takes $O((N+\alpha)^2)$time in the worst case execution, where N shows the total number of requests in each hyper-period of n periodic tasks in the system and α is the number of aperiodic tasks.

2.6 Quality of Service of EDF

The EDF scheduling algorithm is a priority driven algorithm in which higher priority always preempts a lower priority request and is assigned to the request that has earlier deadline, and a higher priority request. This scheduling algorithm is an example of priority driven algorithms with *dynamic priority* assignment in the sense that the priority of a request is

assigned as the request comes. EDF is also known as the *deadline-monotonic* scheduling algorithm. Suppose each time a new ready task arrives, and it is inserted into a ready queue tasks, sorted by their deadlines. If sorted lists are used, the worst case for EDF algorithm takes $O((N+\alpha)^2)$time, where N shows the total number of the requests in each hyper-period of n periodic tasks in the system and is the number of aperiodic tasks.

For the EDF algorithm, we make all the assumptions for the Rate Monotonic algorithm, except that the tasks do not have to be periodic. EDF is called an optimal uniprocessor scheduling algorithm. That is verified by, if EDF cannot feasibly schedule a task set on a uniprocessor, then there is no other scheduling algorithm exists like it. A *time slice swapping* techniques has been taken to prove this. In this technique, we can show that any valid schedule for any task set can be transformed into a valid EDF schedule.

If all tasks are periodic and have relative deadlines equal to their periods, they can be feasibly scheduled by EDF *if and only if* $\sum_{i=1}^{n}\frac{C_i}{P_i} \leq 1$.There is no simple schedulability test corresponding to the case where the relative deadlines are not all equal to the periods; in such a case, we really have to develop a schedule using the EDF algorithm to see if all deadlines are met within a given interval of time. The under given is the schedulability test for EDF under this case.

Define $\cup = \sum_{i=1}^{n} C_i/P_i, D_{max} = \max_{1\leq i\leq n}\{D_i\}$ and P= $lcm(P_1, \ldots . P_n)$, where lcm implies least common multiple. Let h(t) be the sum of the execution times for all tasks whose absolute deadlines are lesser than t. A task set of n is *not* EDF-feasible *if and only if*

-U<1 or

- there exists t<min$\left\{P + D_{max}, \frac{U}{1-U}\max_{1\leq i\leq n}\{P_i - D_i\}\right\}$ such that h(t)>t

Very little is known about algorithms that produce an optimal solution. This is due to either of the following reasons.

- Some real-time scheduling problems are NP-complete. Therefore, we cannot say whether there is any polynomial time algorithm for the problems. For this group, we should go for heuristic algorithms. Let a heuristic algorithm is given, we should investigate for the sufficient conditions for feasible scheduling. The sufficient conditions are used to determine whether a given task set can be scheduled feasibly by the algorithm upon the available processors. Many researchers have also focused on searching for heuristic algorithms whose results are compared to the optimal results.

- In fact, for problems in this class the optimal solution cannot be obtained in polynomial time. Approximation algorithms are polynomial time heuristic algorithms whose performance is compared with the optimal performance.
- As for the second group of real-time scheduling problems, there exists polynomial algorithms which provide feasible schedule of any task set which satisfy some specific conditions. For example any set of periodic tasks which satisfy

  $\sum_{i=1}^{n} {C_i}/{P_i} \leq 1$ is guaranteed to be scheduled feasibly by EDF.We know that an optimal scheduling algorithm is one which may fail to meet a deadline only if no other scheduling algorithm can meet the commitment or deadline. Thus, a feasible scheduling algorithm is optimal if there exists no other feasible algorithm with weak conditions.To prove optimality of a scheduling algorithm, the feasibility conditions of the algorithm should be known. For example there is no dynamic-priority scheduling algorithm that can successfully schedule a set of periodic tasks where $\sum_{i=1}^{n} C_i/P_i > 1$

Therefore, EDF is an optimal algorithm. The optimal algorithm for a real-time scheduling problem is not unique. For instance, in addition to EDF algorithm, there is another optimal dynamic-priority scheduling algorithm, which is the least laxity first (LLF) algorithm. The laxity of a process is defined as the deadline minus remaining computation time. In other words, the laxity of a job is the maximal amount of time that the job can wait and still meet its deadline. The algorithm gives the highest priority to the active job with the smallest laxity. Then the job with the highest priority is executed. While a process is executing, it can be pre-empted by another whose laxity has decreased to below that of the running process. A problem arises with this scheme when two processes have similar laxities. One process will run for a short while and then get preempted by the other and vice versa. Thus, many context switches occur in the lifetime of the processes. The least laxity first algorithm is an optimal scheduling algorithm for systems with periodic real-time tasks **[16, 8, 4]**. If each time a new ready task arrives, it is inserted into a queue of ready tasks, sorted by their laxities. In this case, the worst case time complexity of the LLF algorithm is, $O((N+1)^2)$,where N is the total number of the requests in each hyper-period of periodic tasks in the system and is the number of aperiodic tasks.

CHAPTER 3. PERFORMANCE METRIC RESULTS-COMPARISON OF EDF AND RMS

3.1 Implementation Complexity

When talking about the implementation complexity of a scheduling algorithm, we have to distinguish the case in which the algorithm is developed on top of a generic priority based operating system, from the case in which the algorithm is implemented from scratch, as a basic scheduling mechanism in the kernel.

When considering the development of the scheduling algorithm on top of a kernel based on a set of fixed priority levels, it is indeed true that the EDF implementation is not easy, nor efficient. In fact, even though the kernel allows task priorities to be changed at runtime, mapping dynamic deadlines to priorities cannot always be straightforward, especially when, as common in most commercial kernels, the number of priority levels is small (typically, not greater than 256). For example, consider the case in which two deadlines d_a and d_b are mapped into two adjacent priority levels and a new periodic instance is released with an absolute deadlined_c, such that $d_a < d_c < d_b$. In this situation, there is not a priority level that can be selected to map *dc*, even when the number of active tasks is less than the number of priority levels in the kernel. This problem can only be solved by remapping $d_a$and d_b into two new priority levels which are not consecutive. Notice that, in the worst case, all current deadlines may need to be remapped, increasing the cost of the operation.

If the algorithm is developed from scratch in the kernel using a list for the ready queue, then the only difference between the two approaches is that, while in RM the ready queue is ordered by decreasing fixed priority levels, under EDF it has to be ordered by increasing absolute deadlines. Thus, once the absolute deadline is available in the task control block, the basic kernel operations (e.g., insertion, extraction, get first, dispatch) have the same complexity, both under RM and EDF.

An advantage of RM with respect to EDF is that, if the number of priority levels is not high, the RM algorithm can be implemented more efficiently by splitting the ready queue into several FIFO queues, one for each priority level. In this case, the insertion of a task in the ready queue can be performed in *O*(1). Unfortunately, the same solution cannot be adopted for EDF, because the number of queues would be too large (e.g., equal to 232 if system time is represented by four byte variables).

Another disadvantage of EDF is that absolute deadlines change from a job to the other hand need to be computed at each job activation. Such a runtime overhead is not present under RM, since periods are typically fixed. However, the problem of evaluating the runtime overhead introduced by the two algorithms is more complex.

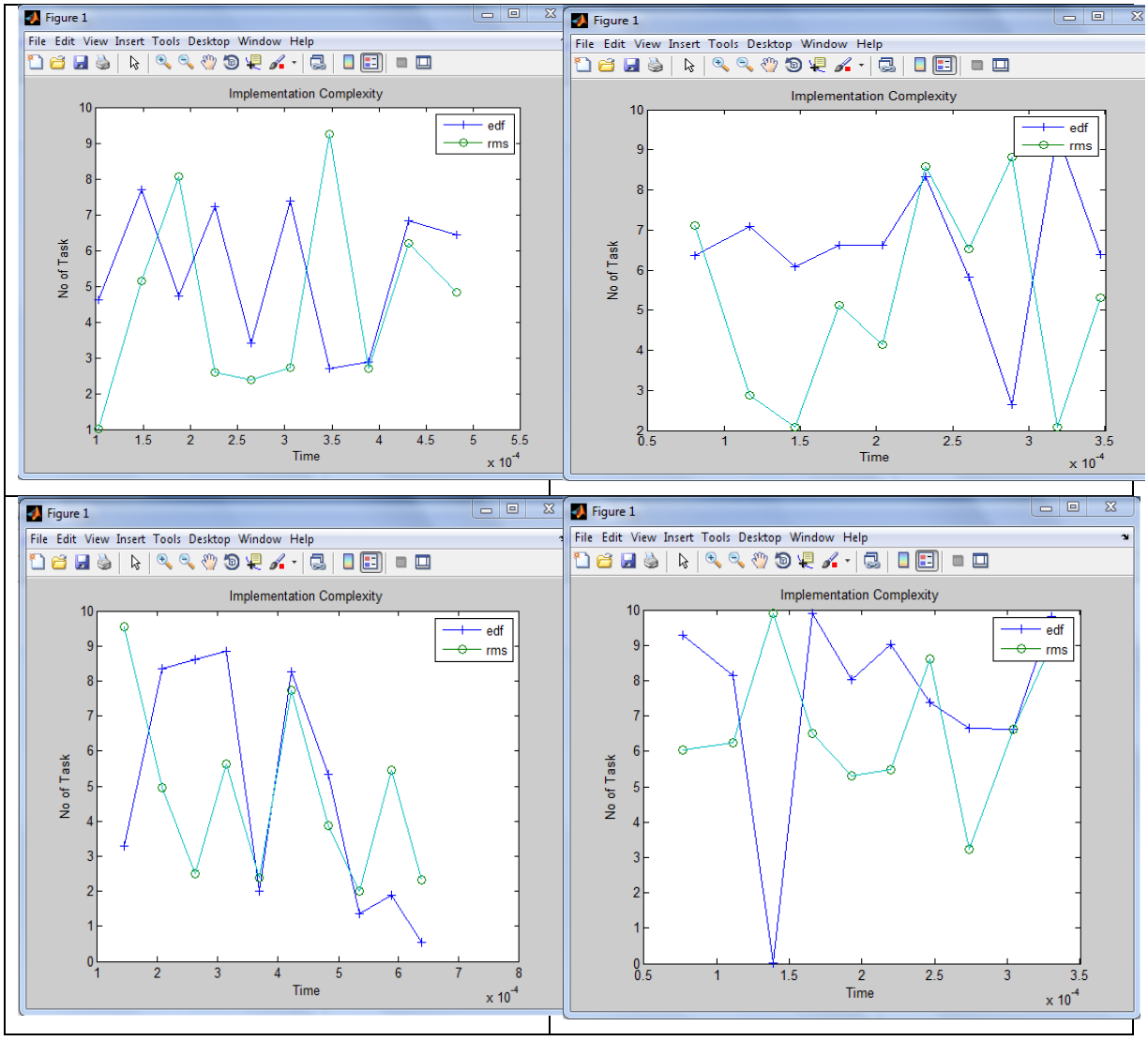

Fig 3.1: We can analyse from the above results that RMS does not support explicit timing constraints on the task set so it's easy to implement and for every new task EDF has to perform a dynamic mapping between absolute deadline and priorities which increases implementation complexity.

3.2 Runtime Overhead

It is commonly believed that EDF introduces a larger runtime overhead than RM, because in EDF absolute deadlines need to be updated from a job to the other, so slightly increasing the time needed to execute the job activation primitive. It is indeed true that, under

EDF, deadlines need to be updated by the kernel at each job activation, because in a periodic task the absolute deadline changes from a job to the other. Whenever the *k*-th job of task τ_i is released at time r_i,k, its absolute deadline has to be computed as $d_i,k = r_i,k + D_i$. Such a computation is not needed under RM, because the priority of task τ_i is assigned based on its period T_i or, if using Deadline Monotonic, based on its relative deadline D_i , which does not change from a job to the other**.[7,11]**

In spite of the extra computation needed for updating the absolute deadline, however, EDF introduces less runtime overhead than RM, when context switches are taken into account. In fact, to enforce the fixed priority order, the number of preemptions that typically occur under RM is much higher than under EDF. For larger task sets the number of preemptions caused by RM increases, thus the overhead due to the context switch time is higher under RM than EDF.

To evaluate the behaviour of the two algorithms with respect to preemptions, a number of simulation experiments have been performed using synthetic task sets with random parameters.

As shown in the plot Fig 3.2(a), each curve has two phases: the number of preemptions occurring in both schedules increases for small task sets and decreases for larger task sets. This can be explained as follows. For small task sets, the number of preemptions increases because the chances for a task to be preempted increase with the number of tasks in the system. As the number of tasks gets higher, however, task execution times get smaller in the average, to keep the total processor utilization constant, hence the chances for a task to be pre-empted reduce. As evident from the graph, such a reduction is much more significant under EDF.

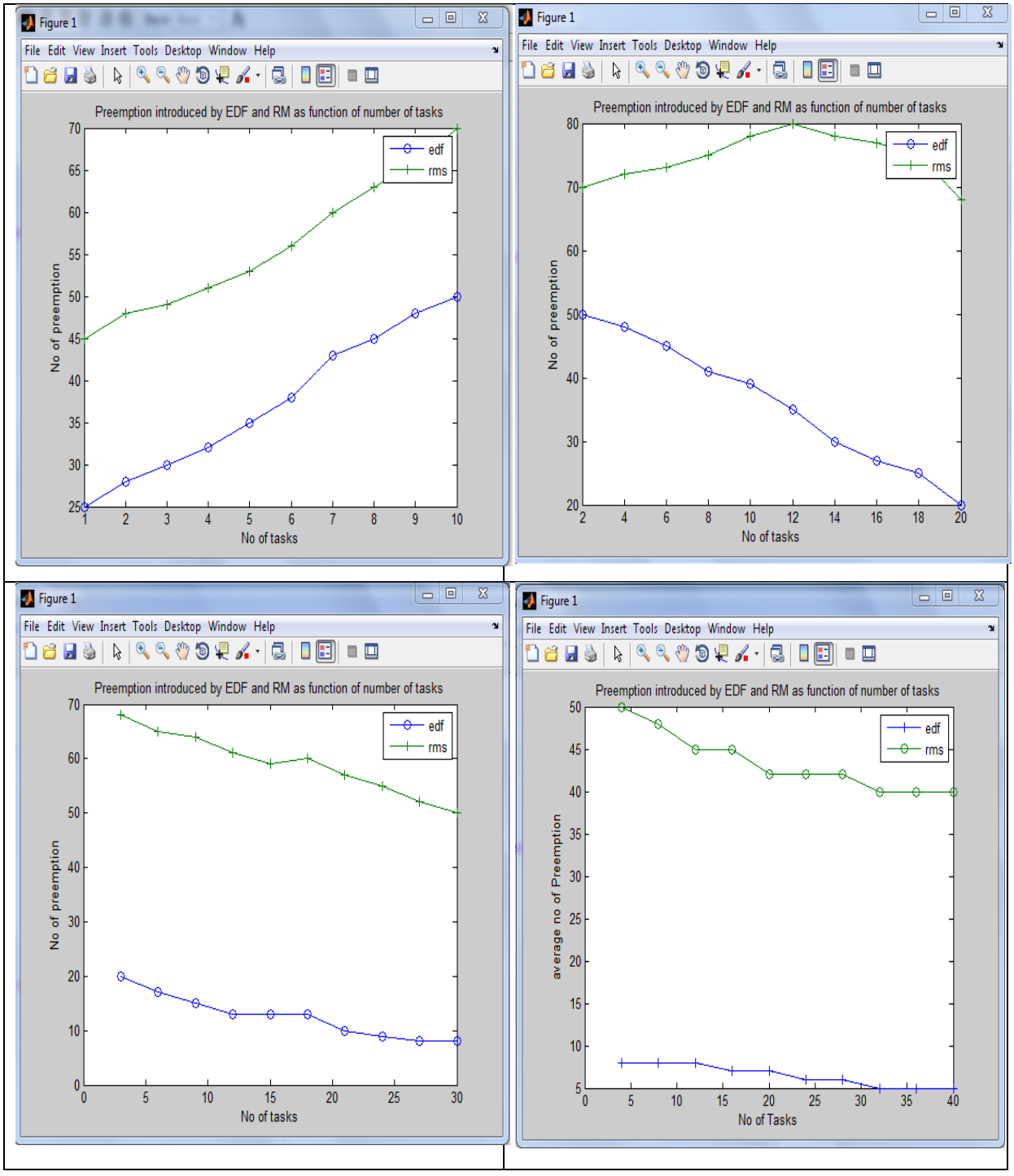

Fig 3.2(a): It shows the average number of preemptions introduced by RM and EDF as a function of the number of tasks. For each point in the graph, the average was computed over 1000 independent simulations, each running for 1000 units of time. In each simulation, periods were generated as random variables with uniform distribution in the range of 10 to 100 units of time, whereas execution times were computed to create a total processor utilization U =0.9.

In another experiment, we tested the behaviour of RM and EDF as a function of the processor load, for a fixed number of tasks.

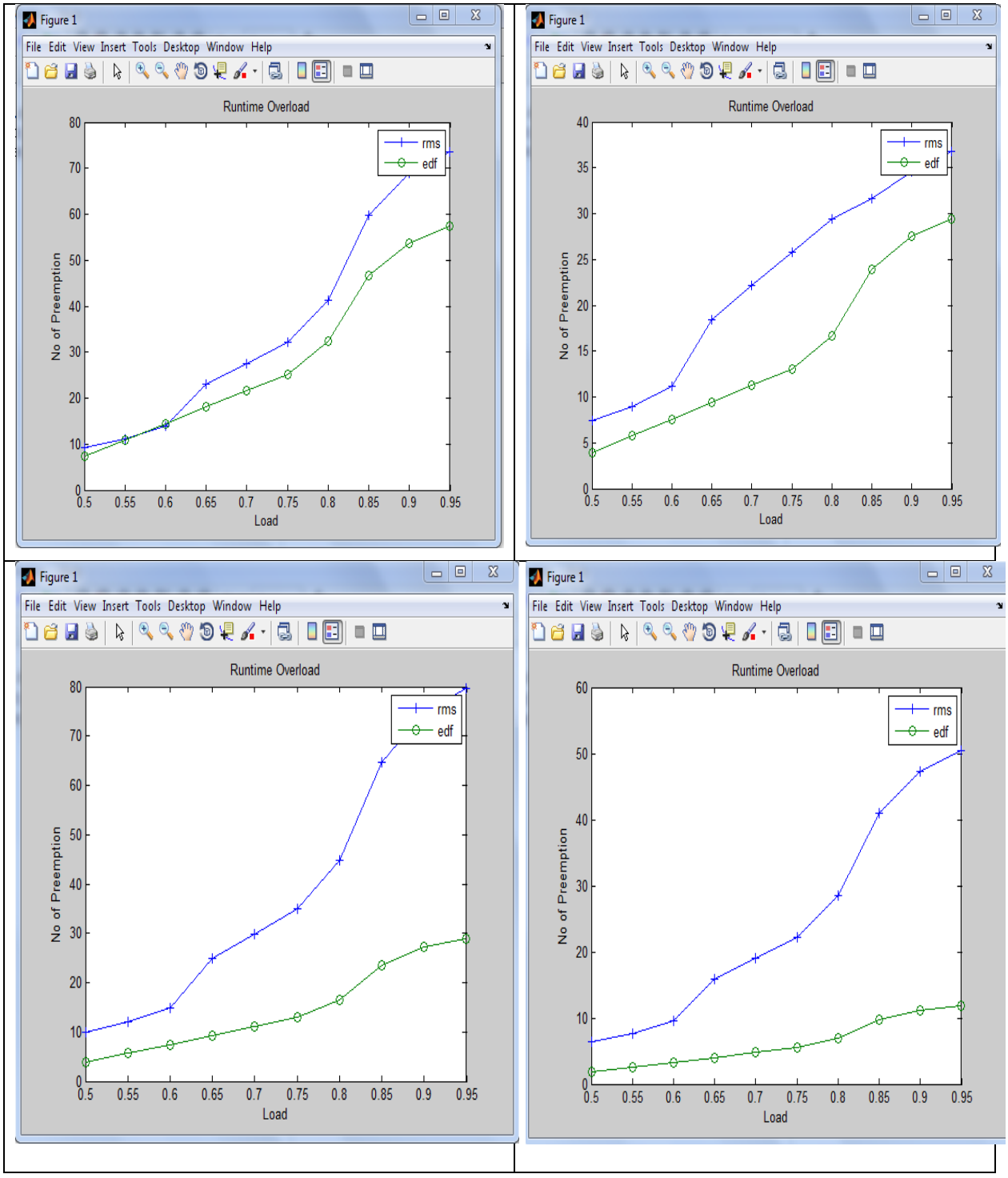

Fig 3.2(b): It shows the average number of preemptions as a function of the load for a set of 10 periodic tasks. Periods and computation times were generated with the same criterion used in the previous experiment, but to create an average load ranging from 0.5 to 0.95.

It is interesting to observe the different behaviour of RM and EDF for high processor load. Under RM, the number of preemptions constantly increases with the load, because tasks with longer execution times have more chances to be preempted by tasks with higher priorities.

Under EDF, however, increasing task execution times does not always imply a higher number of preemptions, because a task with a long period could have an absolute deadline shorter than that of a task with smaller period. In certain situations, an increased execution time can also cause a lower number of preemptions.

This phenomenon is illustrated in the Schedulability Analysis, which shows what happens when the execution time of $\tau 3$ is increased from 4 to 8 units of time. When $C3 = 4$, the second instance of $\tau 2$ is preempted by $\tau 1$, that has a shorter absolute deadline. If $C3 = 8$, however, the longer execution of $\tau 3$ (which has the earliest deadline among the active tasks) pushes $\tau 2$ after the arrival of $\tau 1$, so avoiding its preemption. Clearly, for a higher number of tasks, this situation occurs more frequently, offering more advantage to EDF. Such a phenomenon does not occur under RM, because tasks with small periods always preempt tasks with longer period, independently of their absolute deadlines. **[14]**

The result of the experiment illustrated in above Figure shows that the number of preemptions increases almost linearly with the load under RM, while it decreases for high loads under EDF.

3.3 Schedulability Analysis

The basic schedulability conditions for RM and EDF proposed by Liu and Layland (1973) were derived for a set τ of n periodic tasks under the assumptions that all tasks start simultaneously at time $t = 0$ (that is, $\varphi_i = 0$ for all $i = 1, \ldots, n$), relative deadlines are equal to periods (that is, $di,k = k\ Ti$) and tasks are independent (that is, they do not have resource constraints, nor precedence relations). Under such assumptions, a set of n periodic tasks is schedulable by the RM algorithm if

$$\sum_{i=1}^{n} U_i \leq n\left(2^{1/n} - 1\right) \qquad (1)$$

Under the same assumptions, a set of n periodic tasks is schedulable by the EDF algorithm if and only if

$$\sum_{i=1}^{n} U_i \leq 1 \qquad (2)$$

The schedulability bound of RM is a function of the number of tasks, and it decreases with *n*. We recall that $\lim_{n\to\infty} n(21/n - 1) = \ln 2 \cong 0.69$ meaning that any task set can be scheduled by RM if $U \leq 0.69$, but not all task sets can be scheduled if $0.69 < U \leq 1$. Lehoczky et al. (1989) performed a statistical study and showed that for task sets with randomly generated parameters the RM algorithm is able to feasibly schedule task sets with processor utilization up to about 88%. However, this is only a statistical result and cannot be taken as an absolute bound for performing a precise guarantee test. A more efficient schedulability test, known as the Hyperbolic Bound (HB), was proposed by Bini et al. (2001). This test has the same complexity as the Liu and Layland one, but improves the acceptance ratio up to a limit of $\sqrt{2}$, for large *n*. According to this method, a set of periodic tasks is schedulable by RM if $\prod_{i=1}^{n}(U_i + 1) \leq 2$.

For RM, the schedulability bound improves when periods have harmonic relations. A common misconception, however, is to believe that the schedulability bound becomes 1.00 when the periods are multiple of the smallest period**.[15]**

In the general case, exact schedulability tests for RM yielding to necessary and sufficient conditions have been independently derived by Lehoczky et al. (1989), Audsley et al. (1993), Joseph and Pandya (1986). Using the Response Time Analysis (RTA) proposed in Audsley et al. (2004), a periodic task set (with deadlines less than or equal to periods) is schedulable with the Deadline Monotonic algorithm if and only if the worst-case response time of each task is less than or equal to its relative deadline. The worst-case response time *Ri* of a task can be computed using the following iterative formula:

$$R_i^{(0)} = C_i$$

$$R_i^{(k)} = C_i + \sum_{j:D_j<D_i} \left\lceil \frac{R_i^{(k-1)}}{T_j} \right\rceil C_j$$

where the worst-case response time of task *τi* is given by the smallest value of $R_i^{(k)}$such that $R_i^{(k)} = R_i^{(k-1)}$ .It is worth noting, however, that the complexity of the exact test is pseudo polynomial, thus it is not suited to be used for online admission control in applications with large task sets. To solve this problem, an approximate feasibility test with a tunable complexity has been proposed by Bini and Buttazzo in Bini and Buttazzo (2002). Under EDF, the schedulability analysis of periodic tasks with relative deadlines less than periods can be performed using the Processor Demand Criterion PDC proposed by Baruah et al. (1990). According to this method, a set of tasks is schedulable by EDF if and only if

$$\forall L > 0, \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i \leq L \quad \textbf{[9]}$$

As the response time analysis, this test has also a pseudo-polynomial complexity. It can be shown that the number of points in which the test has to be performed can be significantly restricted to those L equal to deadlines less than a certain value L^*, that is:

$$\forall L \in D,\ D = \{ d_k : d_k < \min(L^*, H)\}$$

where $H = \text{lcm}(T_1, \ldots, T_n)$ is the hyperperiod and $L^* = \frac{\sum_{i=1}^{n} U_i(T_i - D_i)}{1-U}$

In conclusion, if relative deadlines are equal to periods, exact schedulabilty analysis can be performed in $O(n)$ under EDF, whereas is pseudo-polynomial under RM. When relative deadlines are less than periods, the analysis is pseudo-polynomial for both scheduling algorithms, although, in the average, the PDC requires more computational steps.

3.4 Robustness during Overloads

Now we compare the behaviour of RM and EDF during overload conditions that is when the total demand of the task set exceeds the processor capacity. We first consider the two algorithms under permanent overload situations (occurring when $U > 1$), and then under transient overload conditions, caused by sporadic execution overruns in some of the jobs.

a) Permanent Overload

An interesting property of EDF during permanent overloads is that it automatically performs a period rescaling, and tasks start behaving as they were executing at a lower rate. This property has been proved by Cervin et al. (2002) and it is formally stated in the following theorem.

Theorem 1 [Cervin]**.** *Assume a set of n periodic tasks, where each task is described by a fixed period T_i , a fixed execution time C_i , a relative deadline Di , and a release offset $\emptyset_i$ If $U > 1$ and tasks are scheduled by EDF, then, in stationary, the average period $T_i$of each task τ_i is given by $T_i = T_i U$.**[1]***

Notice that under RM, a permanent overload may cause a complete blocking of the lower priority tasks. Let's take three tasks, in fact, generate a total processor workload $U = 4/8+6/12+5/20=1.25$. According to the previous theorem, this means that EDF schedules the tasks as they were executing with periods $T1' = T1U = 10$, $T2' = T2U = 15$, and $T3' = T3U = 25$ (note that $U' = 4/10+6/15+5/25=1$). Indeed, it can be easily verified that in the first interval of 120 units of time, τ_1 executes 12 times (120/10=12), $\tau 2$ executes 8 times (120/15=8), and $\tau 3$ executes almost 5 times (120/25= 4.8).

In conclusion, under permanent overload conditions both the behaviours of RM and EDF are predictable, but, deciding which one is better is highly application dependent.

b) Transient Overload

Another common misconception about RM is to believe that, in the presence of transient overload conditions, deadlines are missed predictably, that is, the first tasks that fail are those with the longest period. Unfortunately, this property does not hold under RM (neither under EDF), and can easily be confuted by the counter example .

Let, there are four periodic tasks with computation times $C1$ =2, $C2$ =3, $C3$ =1, $C4$ =1, and periods $T1$ =5, $T2$ =9, $T3$ =20, $T4$ =30. In normal load conditions, the task set is schedulable by RM. However, if there is a transient overload in the first two instances of task τ_1 (in the example, the jobs have an overrun of 1.5 time units), the task that misses its deadline is not the one with the longest period (i.e.,τ_4), but τ_2.

So we can conclude that, under Rate Monotonic, if the system becomes overloaded, any task can miss its deadline, except the highest priority task independently of its period. Under EDF the situation is not better. The main difference between Rate Monotonic and Earlier Deadline First is that, under RM, an overrun in task τ_i cannot cause tasks with higher priority to miss their deadlines, whereas under EDF any other task could miss its deadline. However, such a property of RM can be of little use if we do not know a priori which task is going to overrun.

The problem caused by sporadic execution overruns can be solved by enforcing temporal isolation among tasks through a resource reservation mechanism in the kernel.

3.5 Jitter and Latency

In a periodic task system which is feasible, the computation calculated by each job should start after its release time and must complete within its approx. deadline. Due to the presence of other concurrent tasks, however, a task may evolve in different ways from instance to instance; that is, the instructions that constitute a job can be shown at different times, relative to its release time, within different tasks. The maximum time variation (relative to the release time) in the occurrence of a particular event in different instances of a task defines the jitter for that event. The jitter of an event of a task τ_i is said to be relative if the variation refers to two

consecutive instances of τ_i, and absolute if it is computed as the maximum variation with respect to all the instances.

For example, the relative response time jitter (RRJ) of a task is the maximum time variation between the response times of any two consecutive jobs. If R_i,k denotes the response time of the k-th job of task τ_i, then the relative response time jitter (RRJ_i)of task τ_i is defined as

$$RRJ_i = \max_k \left| R_{i,k+1} - R_{i,k} \right|$$

Where as the absolute response time jitter (ARJ_i) of task $\tau_i$is defined as

$$ARJ_i = \max_k R_{i,k} - \min_k R_{i,k}$$

In real-time applications, the jitter can be endured when it does not lower the performance of the controlling system. In maximum control applications, a high jitter may cause instability or a jerky behaviour of the controlled system (Marti et al., 2002), hence it must be kept as low as possible .Another misconception about RM is to believe that the fixed priority assignment used in RM reduces the jitter during task implementation, more than EDF.Hence it has been clear that, this is true for the highest priority task, but this property does not hold in any case, as it is shown by example. Let we have a set of three periodic tasks with computation times $C1$ =2, $C2$ =3, $C3$ =2, and periods $T1$ =6, $T2$ =8, $T3$ =12. Under RM, the three tasks experience a response time jitter (both relative and absolute) equal to 0, 2, and 8, respectively. In EDF, the same tasks have a response time jitter (both relative and absolute) equal to 1, 2, and 3, respectively. Therefore, EDF significantly reduces the jitter experienced by task $\tau 3$ by slightly increasing the one of task $\tau 1$.

Clearly, this example does not prove that EDF always introduces less jitter than RM, but just confutes the common belief that RM outperforms EDF in reducing jitter. A specific simulation experiment has been performed to verify the jitter behaviour under the two scheduling algorithms. Ten periodic task set were randomly generated with periods uniformly distributed in [10,200] and fixed total utilization U. The results refer to the Absolute Response-time Jitter (ARJ), which has been normalized with respect to task periods. Hence a value of 1 on task τ_i corresponds to a jitter equal to its period Ti. RM reduces the jitter of high priority tasks at the expenses of tasks with lower priority, EDF treats tasks more evenly, obtaining a significant reduction in the jitter of the tasks with long periods for a small increase in the jitter of tasks with shorter periods.

Another parameter that it is important to minimize in control applications is the input output latency. Assuming that a control task τ_i acquires inputs at the beginning of each instance

and delivers control outputs at the last, the maximum of input–output latency is measured as $L_i = \max_k(f_{i,k} - s_{i,k})$ where $s_{i,k}$ and $f_{i,k}$are the start time and finishing time of job $\tau_{i,k}$respectively.Cervin proved that EDF can always achieve a shorter input–output latency than RM, for any task. This is stated by the following theorem (Cervin, 2003).

Theorem 2 [Cervin]**.** *Given a set of n periodic control tasks performing input at the beginning of each job and output at the end, the maximum input–output latency of each task under EDF is shorter than or equal to the corresponding maximum latency under RM****.[1, 7,9]***

Intuitively, the theorem is true because, under EDF, a task τ_i can never be pre-empted by tasks with a longer relative deadline, but it can be delayed only, if the absolute deadline of the task with longer period is less than the absolute deadline of τ_i. However, such a start time delay does not affect the input–output latency. Moreover, under EDF, the number of preemptions experienced by each job is less than or equal to that experienced under RM.

CHAPTER 4. CONCLUSION AND FUTURE WORK

Here we compared the behaviour of the two most famous policies used today for developing real-time applications: the RM and the EDF algorithm. Although widely used, in fact, there are still many misconceptions about the properties of these two scheduling methods, mainly concerning their implementation complexity, the runtime overhead they introduce, their behaviour during transient overloads, the resulting jitter, and their efficiency to handle aperiodic activities. For all these problems we tried to solve some typical misconception and tried to clarify the properties of the algorithms by illustrating simple examples or reporting formal results from the existing real-time literature. In some cases, specific simulation experiments have also been performed to verify the overhead introduced by context switches, the response time jitter, and the effectiveness in improving aperiodic responsiveness.

Hence we concluded, that the real advantage of Rate Monotonic with respect to Earlier Deadline First is its simpler implementation in commercial kernels that do not provide explicit support for timing constraints, like periods and deadlines. Other properties typically that claimed for Rate Monotonic, are predictability during better jitter control, applied only for the highest priority task, and do not hold for general. On the other hand, EDF allows a full processor utilization, which shows a more efficient and reliable extraction of computational resources and a much better responsiveness of aperiodic activities. These properties become very important for embedded systems working with limited computational resources, and for multimedia systems, where quality of service is controlled through resource reservation mechanisms that are much more efficient under EDF. In fact, most resource reservation algorithms are implemented using service mechanisms same as to aperiodic servers, which are showing better performance under EDF.

Finally, both RM and EDF are not very well suited to work in overload conditions and to achieve jitter control. To cope with overloads, specific extensions have been proposed in the literature, both for aperiodic (Buttazzo and Stankovic, 1995) and periodic (Koren and Shasha, 1995) load. Also a method for jitter control under EDF has been addressed in Baruah et al. (1999) and can be adopted whenever needed.

REFERENCES

[1] Scheduling Algorithms for Real-Time Systems, Arezou Mohammadi and Selim G. Akl, School of Computing, Queen's University, Kingston, Ontario, Canada K7L3N6,2005

[2] G. C. Buttazzo, *"Hard Real-Time Computing Systems: predictable scheduling algorithms and applications,"* Springer company, 2005.

[3] Bini E, Buttazzo GC Schedulability analysis of periodic fixed priority systems. IEEE Trans Compute 53(11):1462–1473 ([5],[6] From A comparative study of rate monotonic schedulability tests Nasro Min-Allah · Samee Ullah Khan ·Nasir Ghani · Juan Li · LizheWang Pascal Bouvry) ,2004.

[4] J. Goossens and P. Richard, *"Overview of real-time scheduling problems,"* Euro Workshop on Project Management and Scheduling, 2004.

[5] T.P Baker Multiprocessor EDF and deadline monotonic schedulability analysis. In processor of the 24th IEEE Real time system Symposium, pages 120-129, Dec 2003

[6] S.Baruah and J Carpenter, Multiprocessor fixed priority scheduling with restricted inter processor migration. In proceedings of the 15th Euromicro Conference on Real Time Systems, page 195-202 IEEE Computer Society Press July 2003.

[7] "Rate monotonic vs. EDF: Judgment Day", Buttazzo, EMSOFT 2003.

[8] Bini E, Buttazzo GC, Buttazzo GM (2001) A hyperplanes bound for the rate monotonic algorithm. In: Proceedings of the 13th euromicro conference on real-time systems, pp 59–67,2001.

[9] S. Schneider, *"Concurrent and Real-time systems, The CSP Approach,"* John Wiley and Sons LTD, 2000

[10] C. M. Krishna and K. G. Shin, *"Real-Time Systems,"* MIT Press and McGraw-Hill Company,1997.

[11] M. Joseph, *"Real-time Systems: Specification, Verification and Analysis,"* Prentice Hall, 1996.

[12] P. A. Laplante, *"Real-time Systems Design and Analysis, An Engineer Handbook,"* IEEE,Computer Society, IEEE Press, 1993

[13] Audsley NC, Burns, A, Richardson, M, Wellings, Applying new scheduling theory to static priority preemptive scheduling. Software Eng J 8(5):284–292,1993.

[14] J. W. de Bakker, C. Huizing, W. P. de Roever and G. Rozenberg, *"Real-Time: Thory in Practice,"* Preceedings of REX Workshop, Mook, The Netherlands, Springer-Verlag company, June 3-7, 1991.

[15] "Algorithm and complexity concerning the preemptive scheduling of periodic, real- time tasks on one processor", Journal of Real-Time Systems, Baruah et al 1990.

[16] Borger, M.W., Klein, M.H, and Veltire, R.A. “Real-Time Software Engineering in Ada.Observations and Guidelines”. Software Engineering Institute Technical Review (1988) .